

# *Kripke – An Sn Transport Mini App*

NECDC 2014

October 22, 2014

Adam J. Kunen



LLNL-PRES-661866

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC



# Overview

- Why another Mini(/Proxy) App?
- What is Kripke?
  - How it relates to ARDRA
  - How it enables research
- Running Kripke
  - Options
  - Test Problems
- Results
- Conclusions

# Why another Mini App?

- None of the existing Co-Design Sn-Transport Mini-Apps are representative of ARDRA
  - SNAP (LANL)
    - Fortran
    - KBA parallel decomposition
    - Sweep contains update of moments
  - UMT (LLNL)
    - C + Fortran
    - Radiation Transport
    - Unstructured
- Refactor of ARDRA
  - Add Concepts: Group Sets, Direction Sets, Zone Sets
  - Possibly Change: Data Striding and Loop Nesting

# Why another Mini App?

- Major Unanswered Questions
  - How does Data Striding and Loop Nesting affect:
    - Memory Performance
      - Bandwidth, Cache Efficiency
    - Parallelism
      - Instruction (SIMD?), Thread (OpenMP, GPU, etc), Task(MPI)
    - What is the interplay with the Platform/Compiler?
  - Investigate New Programming Models
    - RAJA, Kokkos, OCCA, etc.
  - Investigate AMR
    - Load Balancing
    - Partitioning Schemes
    - Sweep performance on imbalanced loads
- So: **We need a simple Mini App that is representative of ARDRA, but also flexible enough to perform exploratory research.**
  - Ardra ~200k lines C/C++
  - Kripke ~2k lines C++

# Simplified “Steady State” Problem

$$\frac{1}{\nu} \frac{\partial \psi}{\partial t} + \Omega \cdot \nabla \psi + \sigma \phi = \sigma_s \psi + \sigma_f \phi + q$$

$H\Psi^{i+1} = L^+ S L \Psi^i + q$

$\downarrow$

$H\Psi^{i+1} = L^+ I L \Psi^i + 0$

$\downarrow$

$H\Psi^{i+1} = L^+ L \Psi^i$

In Kripke:

- $S$  is the identity
- No external sources

# Kripke - A Proxy for ARDRA

$$H\Psi^{i+1} = L^+ L\Psi^i$$

- Sweep Kernel (On Core)
  - 3D Diamond Difference
- Parallel Sweep Algo. (MPI)

- LPlusTimes Kernel  
(moments  $\rightarrow$  discrete)

- LTimes Kernel  
(discrete  $\rightarrow$  moments)

**Not a useful calculation... but representative of ARDRA's computational load**

*3D Jezebel Benchmark Problem: Using ARDRA on rzuseq.llnl.gov (BG/Q)  
12x12x12 zones/task, DD, s16, P4 scattering, 84 groups, spatial decomp, 16 tasks/node*

Kernels	MPI Tasks			
	1	16	64	256
Sweep (On Core + MPI)	35%	40%	47%	48%
LTimes + LPlusTimes	51%	47%	42%	41%
Total (Kripke's Coverage)	86%	87%	89%	89%

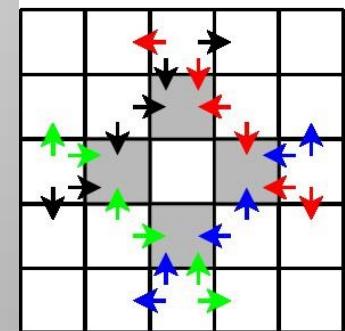
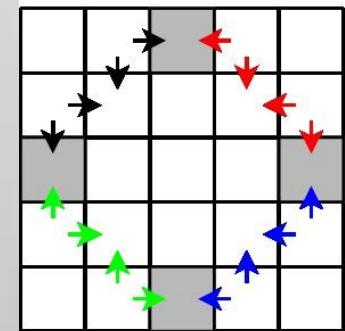
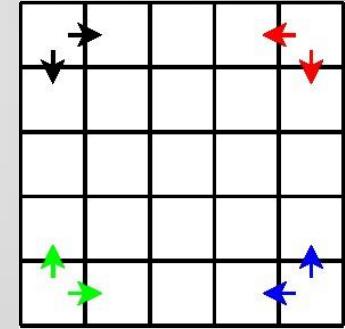
# “ARDRA Solver”

## Solver Iteration:

- Iterate until convergence:
  - Compute RHS
    - LPlusTimes, Scattering, LTimes
  - Run Parallel Sweep Algorithm

## Parallel Sweep Algorithm:

- Foreach Group G:
  - Pipe-Line Directions D:
    - When a given D has upwind dependencies met (either from neighbor or BC):
      - Run Sweep Kernel the local spatial domain for G,D (Sweep Kernel)
      - Send downwind solution to neighbors



Unit of Work: DD Sweep over one Direction, one Group and all local zones

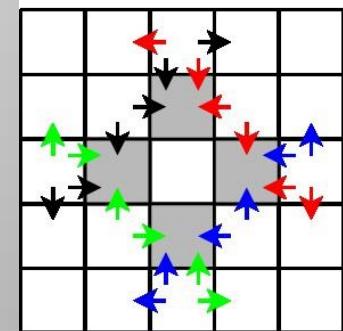
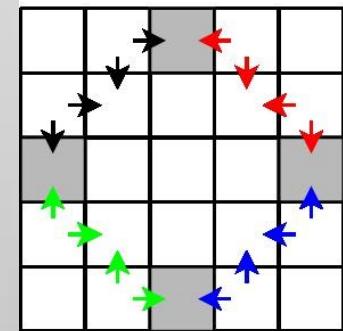
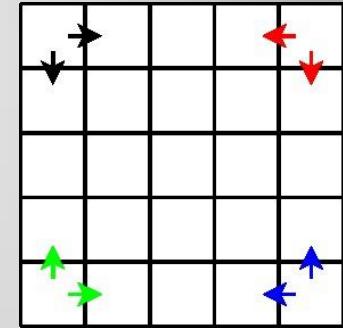
# “Kripke Solver”

## Solver Iteration:

- Iterate **niter** times:
  - Compute RHS
    - LPlusTimes, **Scattering**, LTimes
  - Run Parallel Sweep Algorithm

## Parallel Sweep Algorithm:

- Foreach **GroupSet GS**:
  - Pipe-Line **DirectionSets DS**:
    - When a given **DS** has upwind dependencies met (either from neighbor or BC):
      - Run Sweep Kernel the local spatial domain for all **G,D in GS, DS** (Sweep Kernel)
      - Send downwind solution to neighbors



Unit of Work: DD Sweep a subset of Directions, a subset of Groups and local zones

# ARDRA in a Kripke World

**ARDRA**  
(spatial parallel  
decomp)

*We can model the ARDRA's view  
of the problem in Kripke by forcing  
the unit of work to have 1 direction  
and 1 group.*

**Kripke**

$\Psi[GS][DS][G][D][Z]$

GS = # Group Sets

DS = # Dir. Sets

G = # Groups per GS

D = # Dir. Per DS

("Unit of Work" is in red)

$\Psi[G][D][Z]$

$\Psi[GS][DS][1][1][Z]$

GS = # Ardra Groups

DS = # Ardra Dir.

1 group per GS

1 dir. per DS

# Flexibility of Sets in Kripke

- GroupSet and DirectionSet concepts was borrowed from Texas A&M's code PDT
  - PDT also uses Zone Sets
    - Allows for domain overloading and KBA
    - Future feature of Kripke
- Allows tuning the size of the unit of work:
  - Changes message sizes and # of messages
    - Interplay with parallel sweep performance
  - Allows for more on-node parallelism
    - Take advantage of SIMD and OpenMP
  - Cache performance
- Kernels now act on a 3d index space (GDZ) instead of 1d space (Z) as in ARDRA

**How do we stride the G, D and Z of our unknowns to create the most efficient kernels?**

# “Nestings”

- Kripke keeps the [GS][DS][-][-][-] as the outermost strides, in that order
- Kripke implements all 6 permutations of the data strides for the unit of work:
  - [G][D][Z], [G][Z][D], [D][G][Z],
  - [D][Z][G], [Z][G][D], [Z][D][G]
- We call these “Nestings” as they change the loop nesting of each of the kernels.
- .... and YES, we implement each of the kernels for each of the 6 nestings.

# sweepKernel Psuedocode Example

DGZ Nesting:

$$\Psi[GS][DS][D][G][Z]$$

- Foreach d in D:
  - Foreach g in G:
    - Foreach z in Z:
      - Apply DD operator

ZDG Nesting:

$$\Psi[GS][DS][Z][D][G]$$

- Foreach z in Z:
  - Foreach d in D:
    - Foreach g in G:
      - Apply DD operator

**Loops are re-nested to optimize as much as possible.**

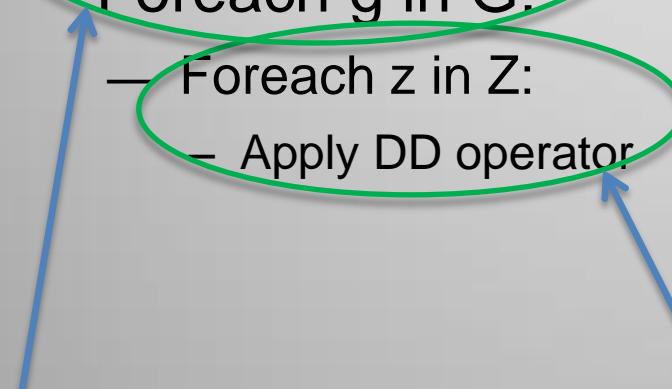
**So what???**

# sweepKernel Psuedocode Example

DGZ Nesting:

$$\Psi[GS][DS][D][G][Z]$$

- Foreach d in D:
  - Foreach g in G:
    - Foreach z in Z:
      - Apply DD operator



ZDG Nesting:

$$\Psi[GS][DS][Z][D][G]$$

- Foreach z in Z:
  - Foreach d in D:
    - Foreach g in G:
      - Apply DD operator

- Sweeps are sequential in nature
- Difficult to thread or get SIMD
  - Hyperplane methods show promise

- Each G and D are independent in the sweep
- Can easily use OpenMP threading here

# sweepKernel Psuedocode Example

DGZ Nesting:

$$\Psi[GS][DS][D][G][Z]$$

- Foreach d in D:
  - Foreach g in G:
    - Foreach z in Z:
      - Apply DD operator
  - Sweeps are sequential in nature
  - Accept that and move on?

ZDG Nesting:

$$\Psi[GS][DS][Z][D][G]$$

- Foreach z in Z:
  - Foreach d in D:
    - Foreach g in G:
      - Apply DD operator

# Nestings? Which one? Oh no!

- Current C/C++ and language abstractions do not:
  - Allow the re-striding of data... NOR
  - Re-nest the loops in a performant way
- In order to investigate how nestings impact performance on different architectures, we must *implement all of them!!!*
- Most codes choose a nesting based on ease of implementation, or other design constraints, not based on performance.
- If we want to refactor our codes to use a specific nesting:
  - Which nesting is the best?
  - How do we choose GS and DS?
  - How do architectures play into this?
  - Kripke will help answer these questions

# Kripkie's Runtime Parameters

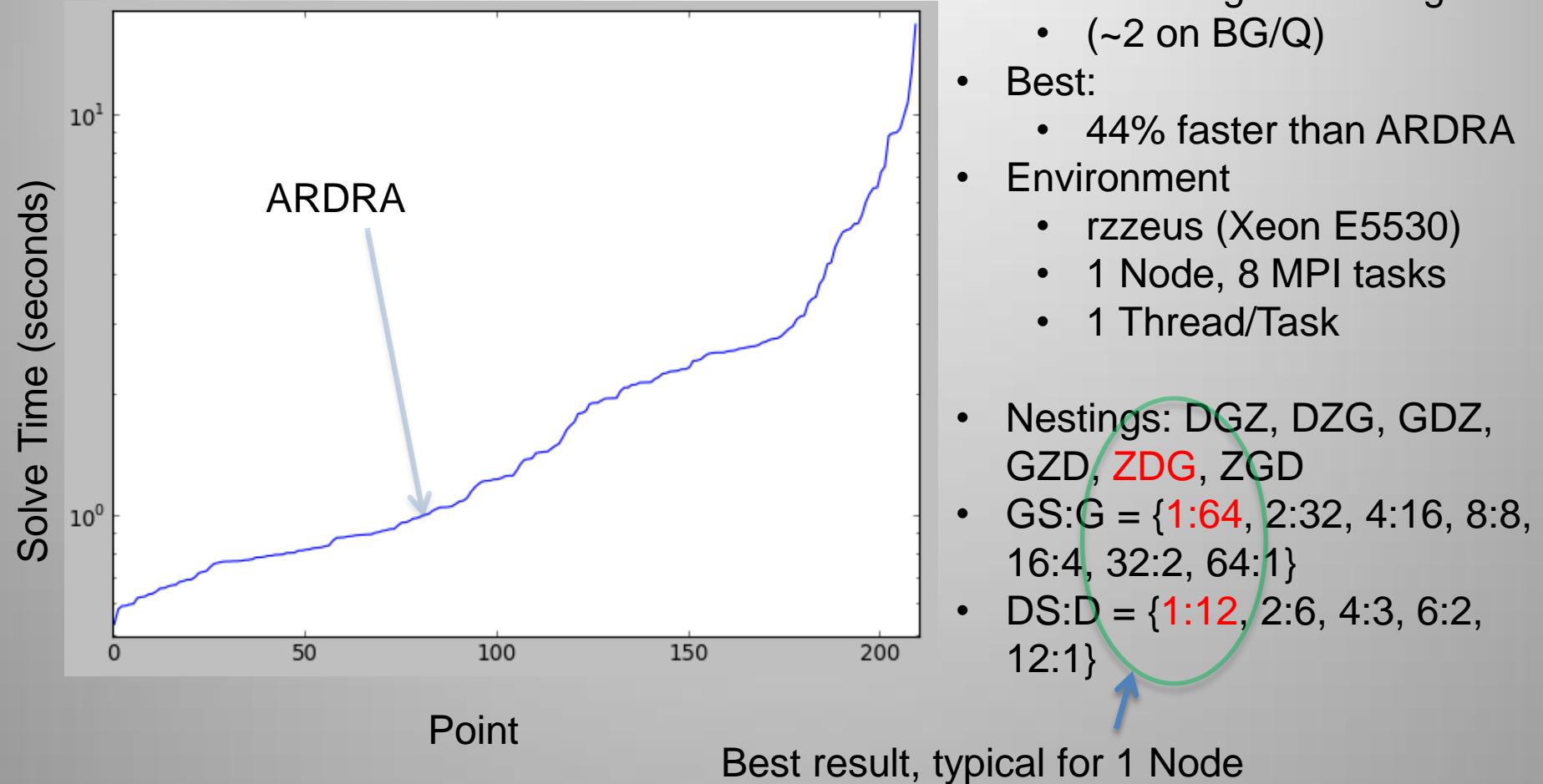
- Command-Line Parameters
  - Number of GS and G-per-set
  - Number of DS-per-octant and D-per-set
  - Nestings (DGZ, DZG, GDZ, GZD, ZDG, ZGD)
  - Number of scattering Legendre moments
    - L, L+ are dimensioned by total directions, and number of moments
  - Total number of zones in X, Y, Z
  - Spatial decomposition in Px, Py, Pz
  - Number of iterations (niter)
- Parameter Space
  - Sets of GS:G, DS:G, and Nestings can be specified
  - Parameter space is product of parameter sets
  - Kripke runs each “point” in the defined parameter space

# Test Problem Definitions

Name	Directions (per Octant)	Groups	Scattering Order	Zones/ Core	Psi (Mb)	Phi (Mb)
KP0	96 (12) ~S8	64	P4	1728 (12x12x12)	81.0	13.5
KP1	256 (32) ~S12	64	P4	1728 (12x12x12)	216.0	13.5
KP2	96 (12)	128	P4	1728 (12x12x12)	162.0	27.0
KP3	96 (12)	64	P9	1728 (12x12x12)	81.0	68.344

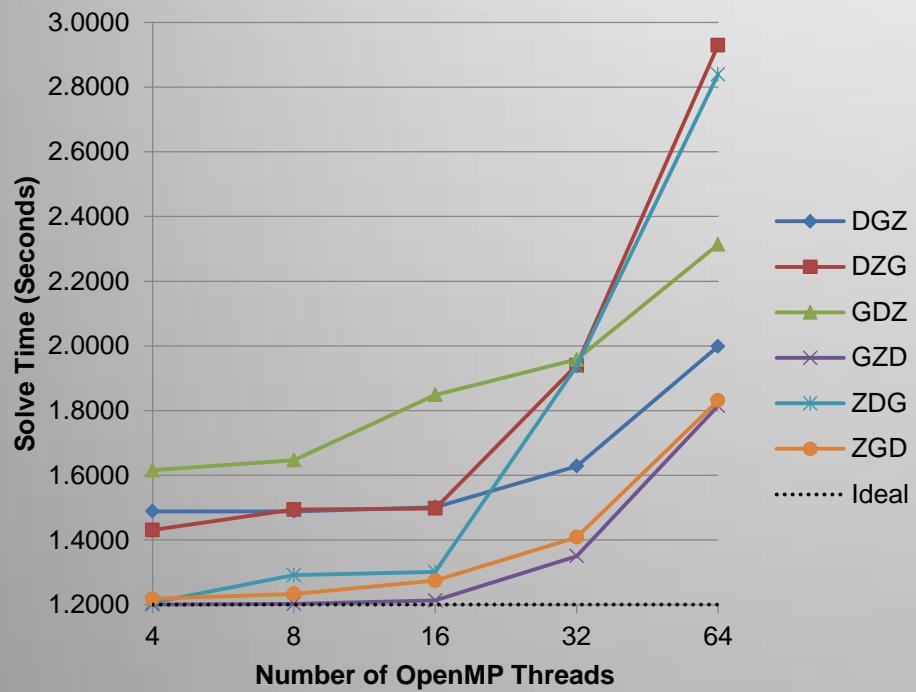
Problems are defined “per-core”, and weak-scaled by increasing zone count.

# Comparison 210 “points” KP0

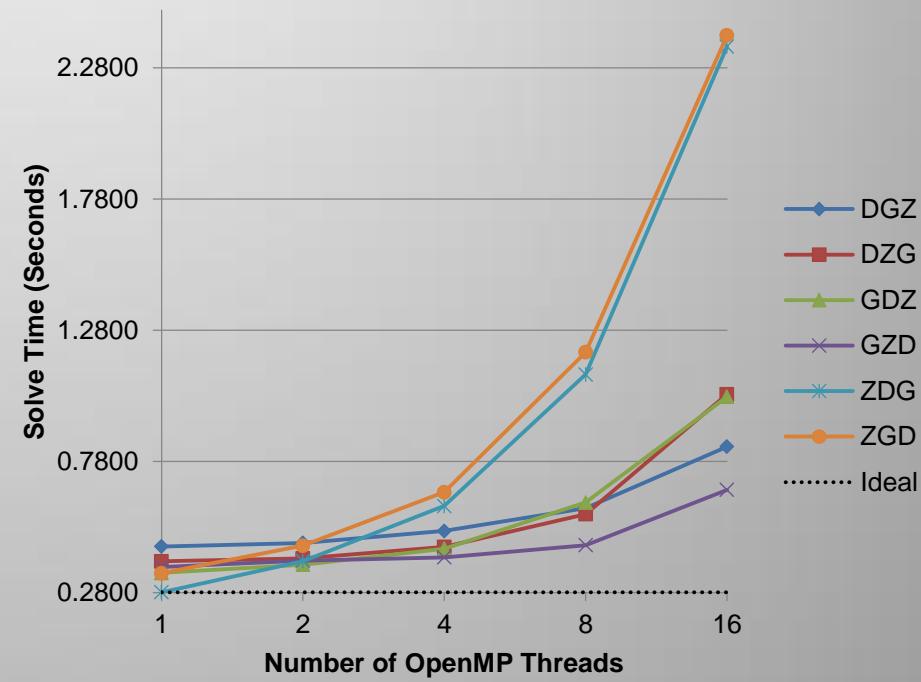


# OpenMP Weak Scaling KP0

OpenMP Weak Scaling  
KP0 -- openmp-1.0 -- Sequoia

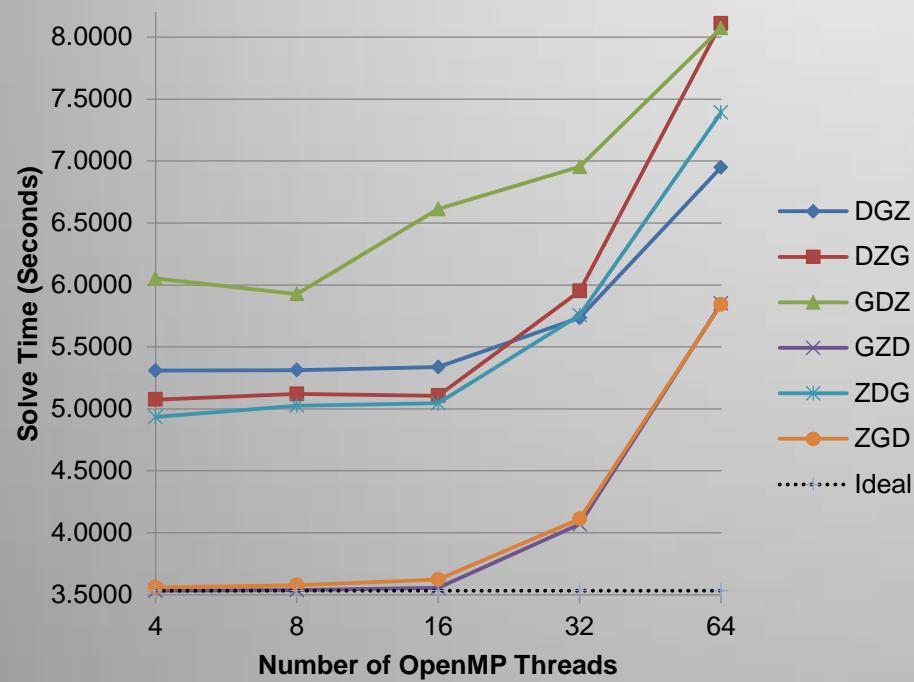


OpenMP Weak Scaling  
KP0 -- openmp-1.0 -- rzmerl

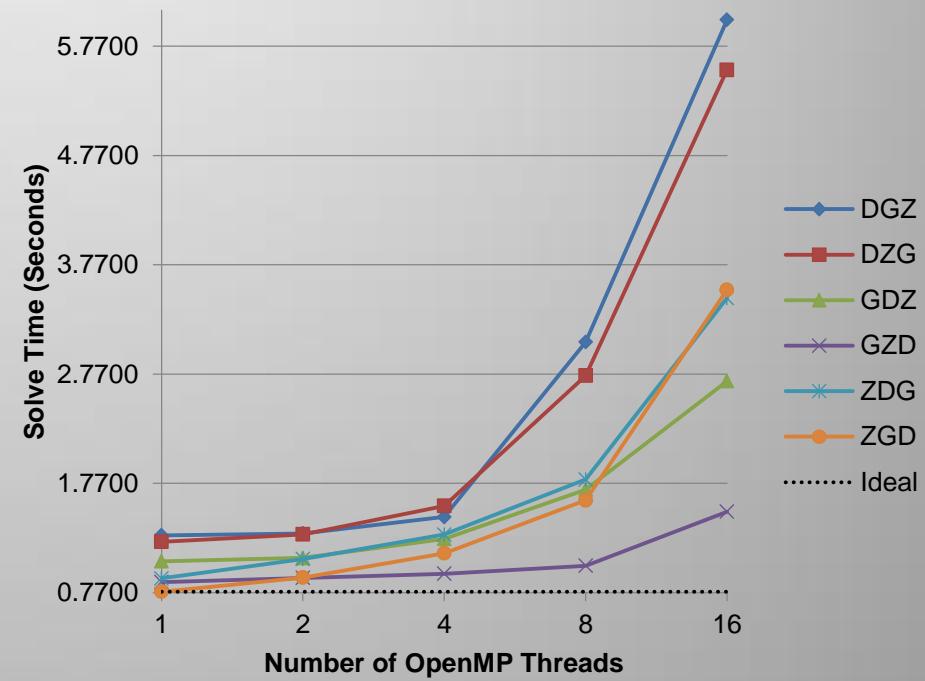


# OpenMP Weak Scaling KP3

OpenMP Weak Scaling  
KP3 -- openmp-1.0 -- Sequoia

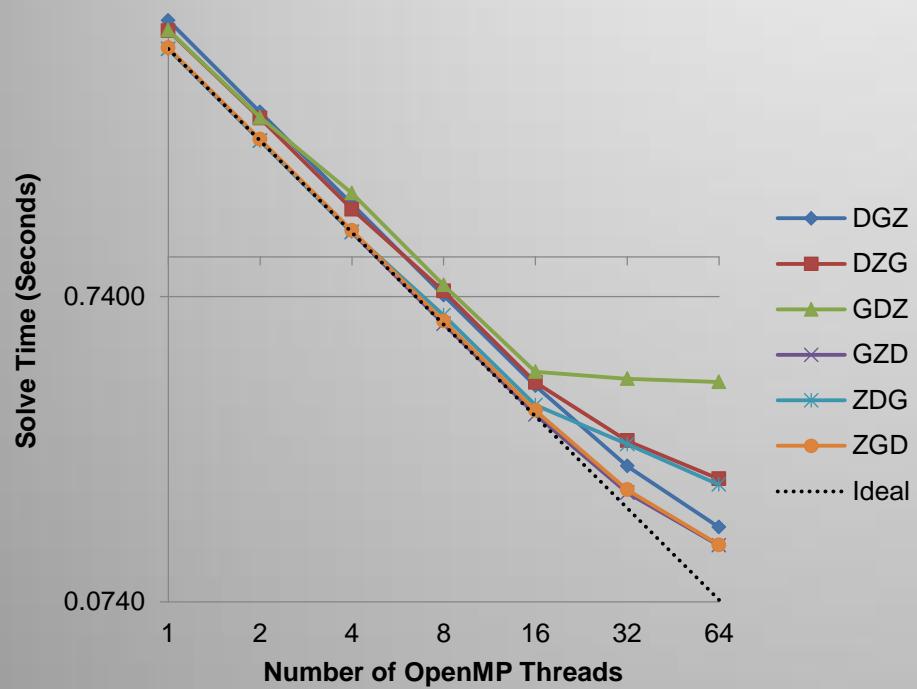


OpenMP Weak Scaling  
KP3 -- openmp-1.0 -- rzmerl

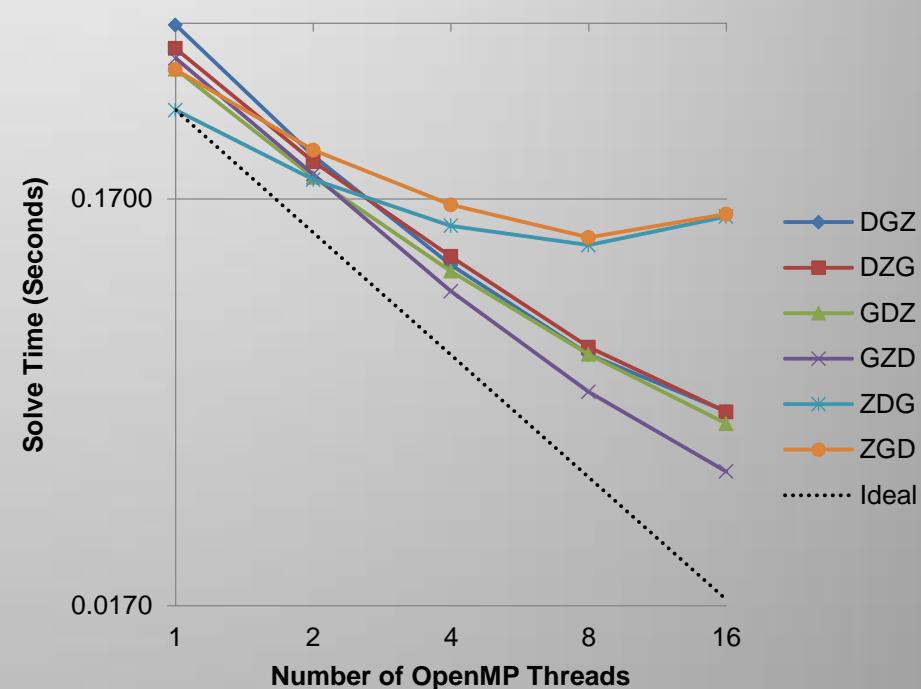


# OpenMP Strong Scaling KP0

OpenMP Strong Scaling  
KP0 -- openmp-1.0 -- Sequoia

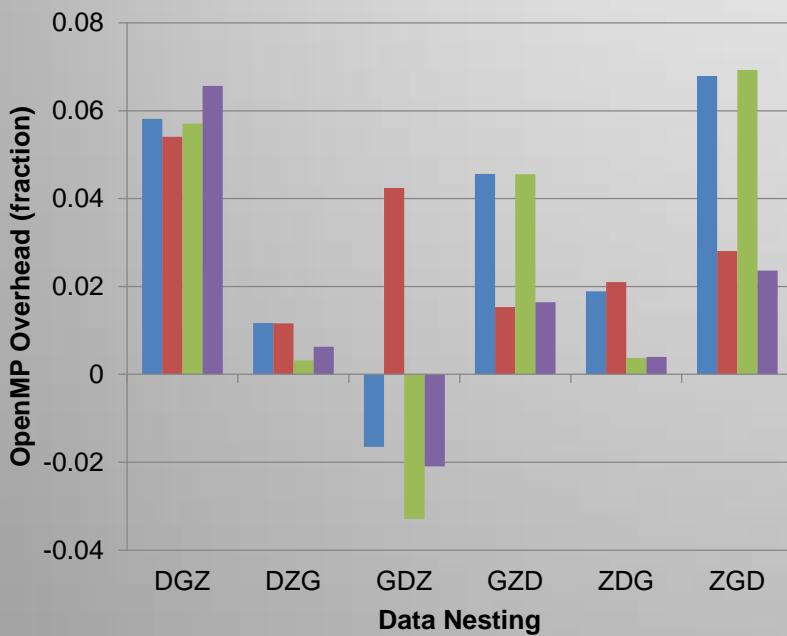


OpenMP Strong Scaling  
KP0 -- openmp-1.0 -- rzmerl

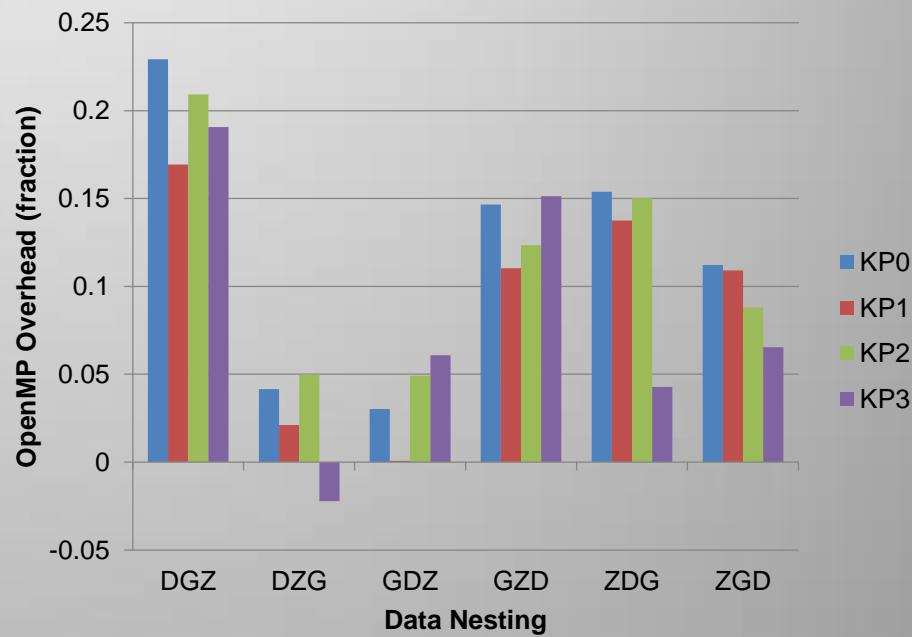


# OpenMP Overhead

OpenMP Overhead on Sequoia  
(XLC 12)  
1-Thread vs. Serial



OpenMP Overhead on rzmerl  
(ICC 14)  
1-Thread vs. Serial



# Conclusion

- Kripke is a new Proxy/Mini app for ARDRA
  - Representative of our current state of the art at LLNL
  - Will provide critical feedback for further Sn development
- Performance is greatly impacted by:
  - Architecture / Operating System
  - Decomposition
  - Data Layout
  - Problem Specification
- Raises More Questions
  - What nesting do we adopt?
    - More than One???
  - How do we choose GS and DS?
    - Possibly use Machine Learning?

# Future Work or Ideas?

- Port to New platforms
  - GPU
    - Cuda?
    - Hyperplane methods?
  - MIC
- AMR Testbed
- Task Graph models for sweeps
- Unstructured Mesh
- Add more kernels to Kripke
  - Scattering kernel
  - 2drz (Cylindrical) Geometry
  - DFEM Spatial Discretizations
- *Anyone interested? Kripke has been released*
  - *Will eventually be available on LLNL CoDesign site*
  - *Contact me: kunen1@llnl.gov*



**Lawrence Livermore  
National Laboratory**